
Parallel
distributed
systems
project

The content

1.INTRODUCTION

The usefulness of distributed systems has spurred a significant amount of research. There have been advances both in hardware and software for the design of distributed software has proven to be more difficult than that of distributed hardware. Architectures, such as Hypercube, provide up to 16k processors connected by a network, but the exploitation of such hardware still remains a challenging task. Our focus in this book will be on techniques for the design of distributed software.

1.1Distributed Systems versus Parallel Systems

Distributed systems offer many advantages over parallel systems. These advantages are as follows:

- Resource sharing: Distributed systems provide resource sharing. For example, an expensive special purpose processor can be shared by multiple organizations.
- Data sharing: Distributed systems provide data sharing in distributed databases. Thus, multiple organizations can share their data with each other.
- Geographical structure: The geographical structure of an application may be inherently distributed. The low communication bandwidth may force local processing. This is especially true for large applications.
- Logical structure: The logical structure of a distributed program may be more than that of a parallel program. A distributed program is more object-oriented since an object can only be accessed through an object message (or a remote procedure call).
- Reliability: Distributed systems are more reliable than parallel systems because the failure of a single computer does not affect the availability of the system.
- Modularity: A distributed system is more flexible because a single processor can be added or deleted easily.
- Low cost: Availability of high bandwidth network and inexpensive workstations also favors distributed computing for economic reasons.

2. Description of the idea

The goal of the project I presented is to create an application using the concept of RMI (Remote Method Invocation) in Java, which allows you to invoke methods from objects located on a remote server as if you were invoking them from a local object. Here's a clearer explanation of the objectives and benefits:

2.1 Distributed Processing:

RMI enables the distribution of tasks and computations across multiple servers. In the code example, you can call a sum function from a client on a different machine than the server that executes the operation.

2.2 Simplified Communication:

RMI provides a simple interface for communication between objects over the network. Developers can invoke operations on remote objects in a similar way to local objects, reducing programming complexity.

2.3 Function Execution Isolation:

Operations and procedures are executed on the server, allowing processing tasks and resources to remain on the server, while the client handles the user interface.

2.4 Easier Network Operations:

RMI handles network details like connections, serialization, and transport operations, allowing developers to focus on the application logic instead of network specifics.

2.5 Code Reusability:

The same interface and server can be used by different clients or applications, making it easier to implement changes or improve applications without the need to rewrite code.

Practical Uses:

RMI can be used in developing enterprise applications, network software, or even in remote control systems where the user needs to control objects or perform operations on a remote server from different clients.

3. Tool Methodology (Flowchart, Pseudocode, or Algorithms)

3.1 describe the working diagram of RMI (Remote Method Invocation)

RMI Working Diagram.

1. Client:

Through the client, the user or application can request an operation (such as invoking a method) on a remote object.

2. Remote Interface:

The interface defines the methods available for remote invocation. This interface sets the methods (e.g., add) that can be called by external clients.

3. Server:

The server contains the implementation of the remote interface. The object that implements this interface is registered in the RMI Registry.

4. RMI Registry:

A service running on the server that registers available remote objects and allows clients to look them up.

5. Method Invocation:

When the client invokes a specific method, the request is sent over the network to the server.

Data serialization is used to convert the request into data that can be transmitted across the network.

The server executes the requested method.

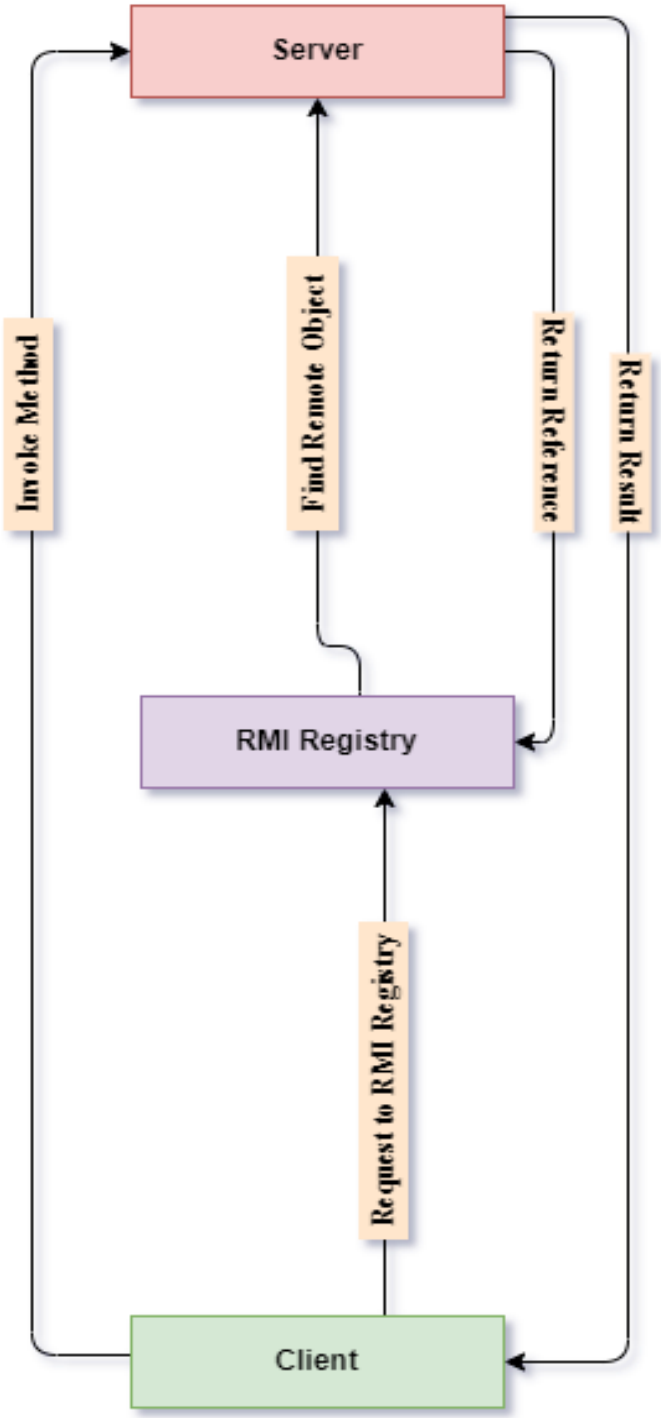
6. Server Response:

After executing the requested function, the server sends the result back to the client using the same serialization process.

7. Receiving the Result on the Client:

The client receives the response and deserializes the data to display the result of the operation.

3.2 The flowchart :



The flowchart illustrates how the client interacts with the server using RMI, and how the server handles multiple tasks using threads.

Steps of the flowchart:

1. The client sends a task to the server via RMI.

2. The server receives the task and creates a new thread to process it.

3. The thread executes the calculations or the task.

4. Once the task is completed, the thread sends the result back to the client via RMI.

5. The client displays the result.

Here is a simplified explanation of the flowchart:

3.3 Pseudocode

This pseudocode illustrates the logic of the program:

3.3.1 Server Pseudocode:

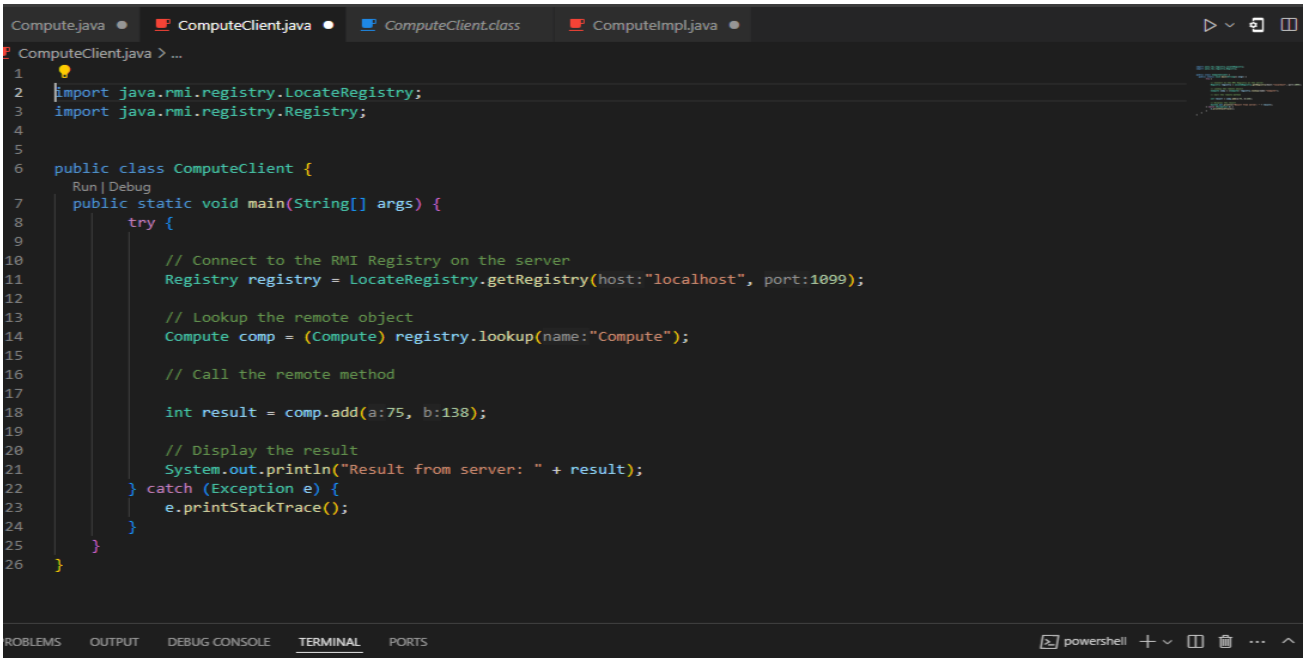
```
Start RMI Registry

Create RemoteObject (ComputeImpl)

Register "Compute" with the RemoteObject in the Registry

When a task is received from the client via RMI

  Create a new thread

  Assign the task to the thread

  Run the thread

    Execute the task (e.g., adding numbers, processing a file, etc.)

    Send the result back to the client via RMI

  Terminate the thread after the task is completed
```

3.3.2Client Pseudocode:

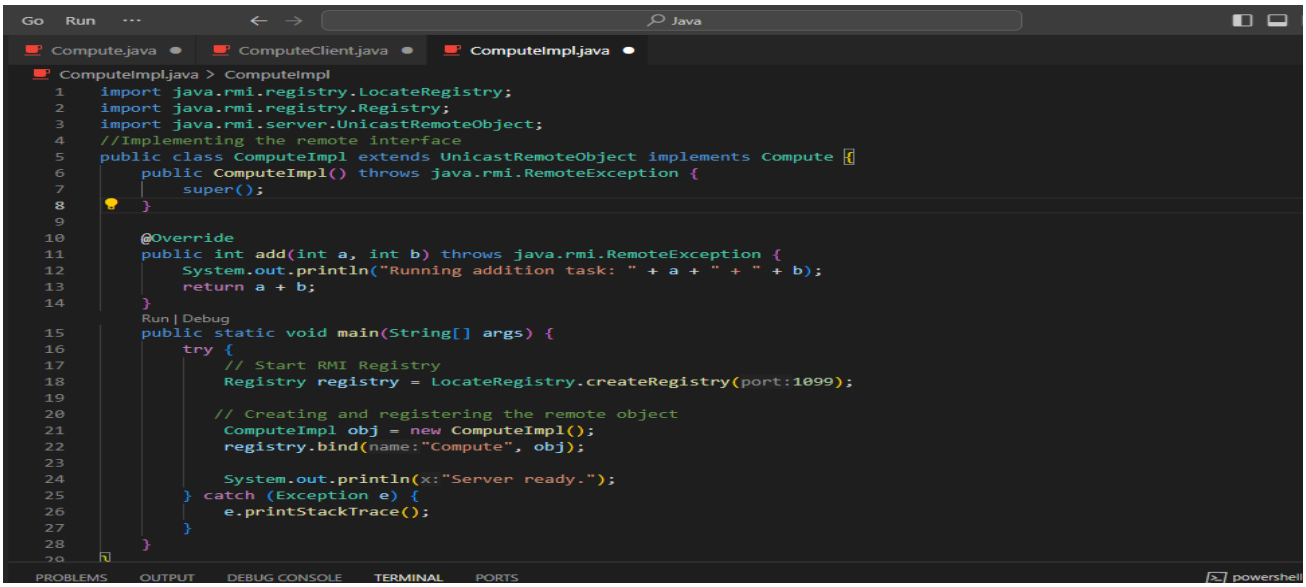
```
Connect to RMI Registry at "server address"

Get RemoteObject "Compute" from the Registry

Call the RemoteObject method add(5, 10)

Receive the result from the server

Display "The result is: " + result
```

4.Implimentation

Code Client :



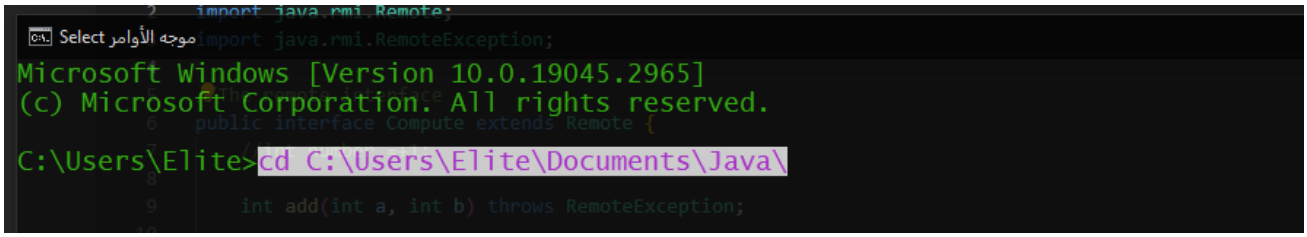
Code Server:



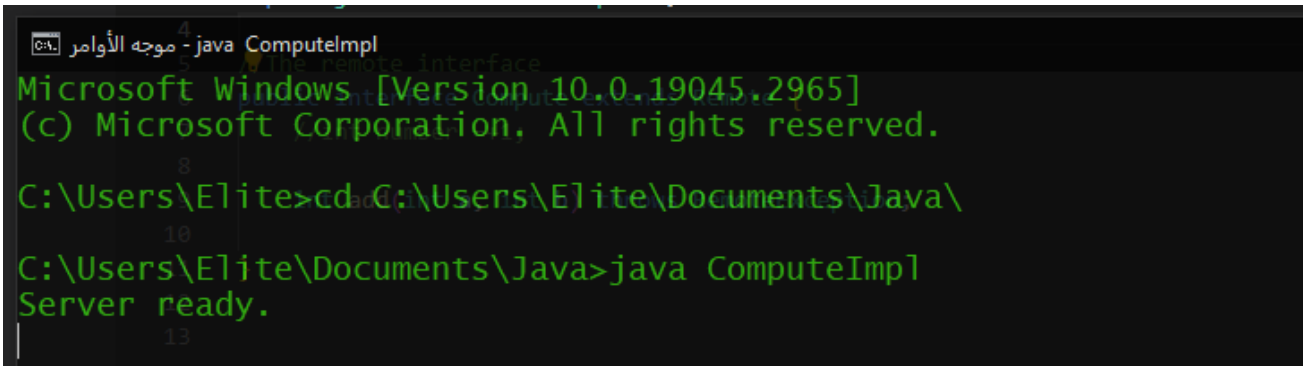
```java
import java.rmi.Remote;
import java.rmi.RemoteException;

// The remote interface
public interface Compute extends Remote {
    //int number =+1;

    int add(int a, int b) throws RemoteException;

}
```

5. Outputs (Files/Screenshot/Results)

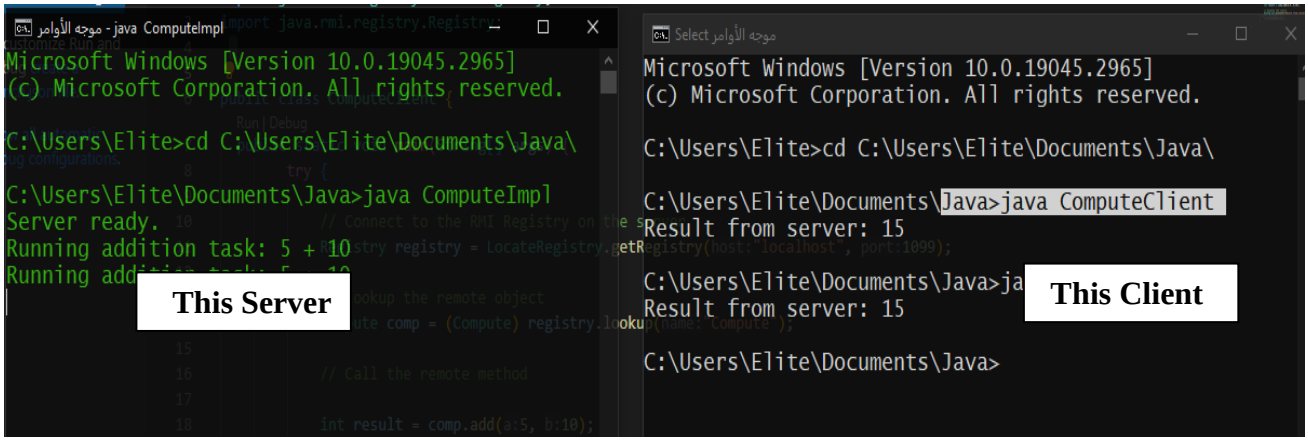
Open folder java via cmd, write this command



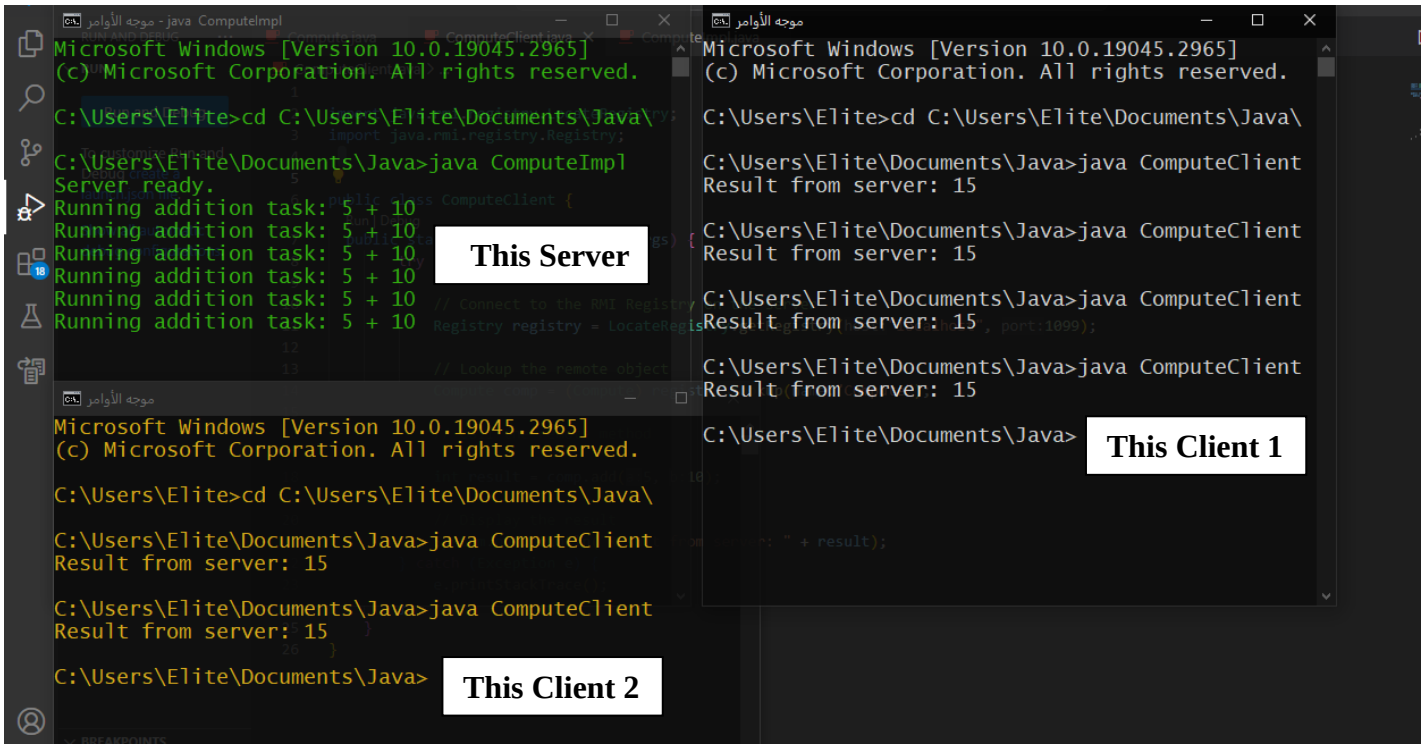
write this command `java ComputeImpl` d for Run the Server



Wishing the Client in Server across Command `java ComputeClient`



- Now we use tow clients to connect to in server and execute the function and bring results.



- The Client 1 execute the function 4 times while, The client 2 execute the function 2 times

5. Conclusion

Overall, the goal is to provide an efficient and secure means of interaction between software on different devices within a network using programming languages like Java. RMI facilitates the execution of remote operations easily and encourages a multi-layered application architecture.

References:

- Garg, Vijay K. *Principles of distributed systems*. Vol. 3144. Springer Science & Business Media, 2012.

- Garg, V. K. (2012). *Principles of distributed systems* (Vol. 3144). Springer Science & Business Media.